苏 州 大 学 实 验 报 告

院、系	计算机学院	年级专业	软件工程	姓名	朱金涛	学号	2327406014
课程名称	操作系统课程实践					成绩	
指导教师	王红玲	同组实验者	无		实验日期	2025 年 10 月 3 日	

实 验 名 称 Linux 内核编译

一. 实验目的

熟悉 Linux 内核编译过程。

二. 实验内容

编译一个 Linux 内核并安装使用。

三. 实验步骤和结果

1. 从官网选择想要安装的内核版本，下载对应的映像

Protocol	Location
HTTP	https://www.kernel.org/pub/
GIT	https://git.kernel.org/
RSYNC	rsync://rsync.kernel.org/pub/

mainline:	6.17	2025-09-28	[tarball]	[pgp]	[patch]	[view diff]	[browse]
stable:	6.16.10	2025-10-02	[tarball]	[pgp]	[patch] [inc. patch]	[view diff]	[browse] [changelog]
longterm:	6.12.50	2025-10-02	[tarball]	[pgp]	[patch] [inc. patch]	[view diff]	[browse] [changelog]
longterm:	6.6.109	2025-10-02	[tarball]	[pgp]	[patch] [inc. patch]	[view diff]	[browse] [changelog]
longterm:	6.1.155	2025-10-02	[tarball]	[pgp]	[patch] [inc. patch]	[view diff]	[browse] [changelog]
longterm:	5.15.194	2025-10-02	[tarball]	[pgp]	[patch] [inc. patch]	[view diff]	[browse] [changelog]
longterm:	5.10.245	2025-10-02	[tarball]	[pgp]	[patch] [inc. patch]	[view diff]	[browse] [changelog]
longterm:	5.4.300	2025-10-02	[tarball]	[pgp]	[patch] [inc. patch]	[view diff]	[browse] [changelog]
linux-next:	next-20251002	2025-10-02					[browse]

```
anders@anders-Legion-Y9000P-IRX8:~$ uname -r
6.8.0-84-generic
```

实验时，我当前内核版本为 **6.8.0**，考虑到固件驱动兼容性的问题，所以我选择了一个与之距离相近的内核版本：**6.9.1**。

2. 解压缩 Linux 内核源码

源码包 linux-6.9.1.tar.xz 采用 xz 压缩，用以下命令解压：

tar xJf linux-6.9.1.tar.xz

3. 配置编译选项

映像既提供 menuconfig 这样的图形化界面的配置，也有.config 文件直接配置。

本次实验为确保内核更新后能成功适配本电脑，我选择了较为稳妥的方式配置，即先将旧的.config 文件 cp 到 6.9.1 目录下，再对.config 文件做进一步微调。

通过 nano 打开.config 进行编辑，完成了选做部分的任务（把版本号后面的 8 改成你学号后 3 位），同时为了提高辨识度，在版本尾部加上了我名字的拼音。**（学号：014，名字：JinTao）**

```
# SPDX-License-Identifier: GPL-2.0
VERSION = 6
PATCHLEVEL = 9
SUBLEVEL = 014
EXTRAVERSION = Jintao
NAME = Hurr durr I'ma ninja sloth
```

接下来输入命令：make kernelversion 6.9.014Jintao 来**配置新内核**

```
  HOSTCC  scripts/kconfig/symbol.o
  HOSTCC  scripts/kconfig/util.o
  HOSTLD  scripts/kconfig/conf
.config:10814:warning: symbol value 'm' invalid for ANDROID_BINDER_IP
C
.config:10815:warning: symbol value 'm' invalid for ANDROID_BINDERFS
*
* Restart config...
*
*
* Processor type and features
*
Symmetric multi-processing support (SMP) [Y/n/?] y
Support x2apic (X86_X2APIC) [Y/?] y
Enable MPS table (X86_MPPARSE) [Y/n/?] y
x86 CPU resource control support (X86_CPU_RESCTRL) [Y/n/?] y
Flexible Return and Event Delivery (X86_FRED) [N/y/?] (NEW) y
```

遇到询问（y/n）**直接按回车键**选择默认项即可。

4. 编译内核

考虑到内核编译可能需要较长时间，而我的 Linux 系统并不是虚拟机，而是通过**移动固态硬盘安装的双系统**。所以完全可以把电脑的资源充分发挥起来，通过 nproc 命令查看电脑的可用处理器核心数量。

```
anders@anders-Legion-Y9000P-IRX8:/usr/src/linux-6.9.1$ nproc
32
```

显示为 32，这时候其实可以用到-j64。

为了进一步理解内核编译的过程，本实验并未直接采用 make 命令，而是拆分成**编译内核（make vmlinux）和压缩映像（make bzImage）：**

1) 编译内核：

执行 sudo make -j64 vmlinux，等待三分钟左右即可编译完成，会在代码录下生成 vmlinux 文件：

```
anders@anders-Legion-Y9000P-IRX8:/usr/src/linux-6.9.1$ ll vm* -h
-rwxr-xr-x 1 root root  396M 10月  1 21:56 vmlinux*
-rw-r--r-- 1 root root  262K 10月  1 21:56 vmlinux.a
-rw-r--r-- 1 root root   15M 10月  1 21:56 vmlinux.map
-rw-r--r-- 1 root root 1001M 10月  1 21:56 vmlinux.o
-rw-r--r-- 1 root root  766K 10月  1 21:56 vmlinux.symvers
```

2) 压缩映像：

执行 sudo make -j64 bzImage 命令，在 arch/x86/boot/下得到压缩好的 bzImage 文件：

```
anders@anders-Legion-Y9000P-IRX8:/usr/src/linux-6.9.1$ ll -h arch/x86/boot/bzImage
-rw-r--r-- 1 root root 13M 10月  1 21:58 arch/x86/boot/bzImage
```

可以看出，364MB 的内核被压缩到了 13MB，文件小了非常多。这一方面是压缩算法，另一方面也是因为原来的文件包含了所有的内核符号和调试信息，而压缩后的文件删除了这些信息。这样，压缩映像 bzImag 能够在保持启动所需功能的同时，大幅度减小文件大小，从而加快加载速度并减少内存占用。

5. 编译模块

在 Linux 系统内核的编译安装过程中，make modules 命令的核心作用是编译内核模块，具体功能如下：

- 编译内核模块代码 Linux 内核采用“核心 + 模块”的架构，其中大量硬件驱动、文件系统支持、网络协议等功能以“模块（Module）”形式存在（而非直接编译到内核镜像中）。这些模块的源代码位于内核源码目录的 drivers/、fs/、net/ 等子目录中。make modules 会根据当前的内核配置文件（.config），对所有被配置为“模块化编译（=m）”的功能代码进行编译，生成对应的 .ko（Kernel Object）二进制文件。
- 与内核核心保持一致性模块编译时会严格依赖当前内核的配置参数、版本信息和核心代码编译结果，确保生成的 .ko 文件与最终编译出的内核镜像（vmlinuz）在接口、数据结构等方面完全兼容，避免模块加载时出现版本不匹配或功能冲突。
- 为后续安装做准备编译生成的 .ko 文件会暂时存放在源码目录的对应子文件夹中，后续通过 make modules_install 命令可将这些模块复制到系统的标准路径（通常为 /lib/modules/<内核版本>/），使内核在启动后能动态加载这些模块（通过 modprobe 或 insmod 命令）。

6. 安装模块

执行 make modules_install 将编译好的模块安装到系统的模块目录，一般是/lib/modules/$(uname -r)。这样，**系统可以为不同版本的内核保持不同版本的模块**，如下图所示：

```
anders@anders-Legion-Y9000P-IRX8:/usr/src/linux-6.9.1$ ll /lib/modules
总计 40
drwxr-xr-x   8 root root  4096  9月 29 14:39 ./
drwxr-xr-x 137 root root 12288  9月 29 14:49 ../
drwxr-xr-x   2 root root  4096  7月 18 16:47 6.8.0-40-generic/
drwxr-xr-x   2 root root  4096  9月  8 14:40 6.8.0-60-generic/
drwxr-xr-x   2 root root  4096  9月 20 15:16 6.8.0-64-generic/
drwxr-xr-x   2 root root  4096  9月 29 14:49 6.8.0-79-generic/
drwxr-xr-x   5 root root  4096  9月 20 15:15 6.8.0-83-generic/
drwxr-xr-x   5 root root  4096  9月 29 14:39 6.8.0-84-generic/
```

为了确保内核与模块的完整性和一致性，选择了**先安装模块后安装内核**。模块作为内核的扩展功能（如驱动、文件系统等），需提前被复制到系统指定路径（/lib/modules/<内核版本>/），而后续安装内核时生成的引导配置（如 GRUB 配置）会依赖这些已安装的模块信息。若先安装内核再安装模块，可能导致引导配置缺失模块路径关联，或内核启动时因找不到对应的 .ko 文件而无法加载必要功能，最终引发启动失败或硬件 / 服务异常。因此，先安装模块再安装内核，是保证两者版本匹配、路径正确关联的必要顺序。

7. 安装内核

make install 命令是 Linux 内核编译安装流程的最后一步，其核心作用是将编译完成的内核相关文件部署到系统指定位置，并配置引导程序以支持新内核启动。结合具体步骤如下：

- 复制内核镜像文件到系统引导目录首先，将编译生成的内核镜像（通常为 vmlinuz-<内核版本>，是经过压缩的可执行内核代码）复制到 /boot 目录。这一目录是系统启动时 BIOS/UEFI 引导程序查找内核文件的标准路径，确保引导程序能识别并加载新内核。
- 安装内核符号表文件同时，将内核符号表文件 System.map-<内核版本> 复制到 /boot 目录。该文件记录了内核中函数、变量的内存地址映射，是内核调试、模块加载（确保模块与内核符号匹配）以及系统日志分析的重要依据，缺失会导致模块加载失败或调试信息混乱。
- 生成初始内存磁盘（initramfs/initrd）命令会调用 mkinitramfs 或 mkinitrd 工具，基于新内核版本和已安装的模块（/lib/modules/<内核版本>/ 下的 .ko 文件）生成初始内存磁盘镜像（如 initrd.img-<内核版本> 或 initramfs-<内核版本>.img），并存放于 /boot 目录。这一镜像包含了内核启动初期所需的关键驱动（如磁盘控制器、文件系统驱动）和初始化脚本，确保内核在访问真正的根文件系统前能完成硬件初始化，是现代 Linux 系统启动的必要组件。
- 更新引导加载程序配置最后，make install 会触发系统引导程序（如 GRUB2）的配置更新（例如执行 update-grub 或 grub-mkconfig）。这一步会扫描 /boot 目录中的新内核文件和初始内存磁盘，将新内核添加到引导菜单中，使系统重启时能通过引导程序选择加载新内核。

8. 重启系统，检查效果

sudo reboot 之后进入终端查看 uname -r，发现内核已成功安装：

```
anders@anders-Legion-Y9000P-IRX8:~$ uname -r
6.9.014Jintao
```

实验完成！

四. 实验总结

问题解决： 在编译内核过程中遇到了一个问题

```
anders@anders-Legion-Y9000P-IRX8:/usr/src/linux-6.9.1$ sudo make -j64 vmlinux
[sudo] anders 的密码：
mkdir -p /usr/src/linux-6.9.1/tools/objtool && make O=/usr/src/linux-6.9.1 subdi
r=tools/objtool --no-print-directory -C objtool
  INSTALL libsubcmd_headers
  CALL    scripts/checksyscalls.sh
make[3]: *** 没有规则可制作目标“debian/canonical-certs.pem”，由“certs/x509_certi
ficate_list” 需求。 停止。
make[2]: *** [scripts/Makefile.build:485：certs] 错误 2
make[2]: *** 正在等待未完成的任务....
make[1]: *** [/usr/src/linux-6.9.1/Makefile:1919：.] 错误 2
make: *** [Makefile:240：__sub-make] 错误 2
```

查询后发现起因是从官网下载新内核时没有连同证书一起下载，他们是分离开的。这就导致编译时需要加载证书文件而找不到，所以报错。而内核证书的核心作用是通过签名与验证机制，确保加载的内核模块来自可信来源、未被篡改，从而维护内核安全与系统完整性，同时支持 UEFI 安全启动等高级安全功能；而个人使用场景中，一方面日常需求多依赖系统默认的官方签名模块，极少需要自定义模块，无需证书也能满足使用，另一方面跳过证书配置可简化操作流程，且个人设备面临的恶意模块风险较低，基本安全需求可通过不随意安装未知模块来保障，因此通常不需要配置内核证书。

所以我的解决方案是修改.config 文件中关于内核证书的调用：

```
CONFIG_MODULE_SIG_KEY=""
CONFIG_MODULE_SIG_KEY_TYPE_RSA=y
# CONFIG_MODULE_SIG_KEY_TYPE_ECDSA is not set
CONFIG_SYSTEM_TRUSTED_KEYRING=y
CONFIG_SYSTEM_TRUSTED_KEYS=""
CONFIG_SYSTEM_EXTRA_CERTIFICATE=y
CONFIG_SYSTEM_EXTRA_CERTIFICATE_SIZE=4096
CONFIG_SECONDARY_TRUSTED_KEYRING=y
```

```
CONFIG_MODVERSIONS=y
CONFIG_ASM_MODVERSIONS=y
CONFIG_MODULE_SRCVERSION_ALL=y
CONFIG_MODULE_SIG=n
```

```
CONFIG_SYSTEM_REVOCATION_KEYS=""
```

遇到一个很严重的问题：

当我在双系统（移动固态硬盘中的 Linux）内核编译这一套全部结束后，重启电脑。发现连不上网了，网络连接符号一直在闪烁。多方查询均未果，最终都没有解决这个问题。疑似是网卡驱动在编译过程中出错了，或者就是版本不适配。

然后我就把我硬盘中的 Linux 内核退回去了，不敢在上面做了。在 windows 下装了虚拟机重新进行实验，取得成功，一切正常。

后续实验均在虚拟机中完成。

原：6.14，现：6.17

```
tt@tt-VMware-Virtual-Platform:~$ uname -r
6.17.014Jintao
```

实验感悟：

本次 Linux 内核编译实验让我完整经历了从源码到可运行系统的内核构建全过程，不仅掌握了内核版本选择、配置调整、模块编译与安装的核心技术环节，更深刻理解了 Linux 内核的模块化架构设计与启动机制。通过自主解决内核证书缺失等问题，我学会了分析编译错误根源并灵活调整配置方案，提升了实际问题解决能力。实验还让我认识到内核压缩与模块管理的精妙设计——如何在保持功能完整性与提升启动效率间取得平衡。这次实践加深了我对操作系统底层工作原理的理解，为后续深入探索系统编程与内核开发奠定了扎实基础。